I Erläuterungen

Voraussetzungen gemäß KCBG und Abiturerlassen BG jeweils in der für den Abiturjahrgang geltenden Fassung

Standardbezug
Die nachfolgend ausgewiesenen Kompetenzbereiche sind für die Bearbeitung der jeweiligen Aufgabe besonders bedeutsam. Darüber hinaus können weitere, hier nicht explizit ausgewiesene Kompetenzen für die Bearbeitung der Aufgabe nachrangig bedeutsam sein, zumal die Kompetenzen in engem Bezug zueinander stehen. Die Operationalisierung des Bezugs zu den Kompetenzbereichen des Standardbezugs erfolgt in Abschnitt II.

Aufgabe	Kompetenzen				
	K1	K2	K3	K4	K5
1.1			X		
1.2	X	X			
1.3		X	X		
1.4				X	
1.5			X		
1.6.1	X	X			
1.6.2		X		X	
1.6.3			X		
1.7.1	X	X			
1.7.2			X		
2.1		X			
2.2.1		X			
2.2.2				X	
2.2.3				X	
2.2.4				X	
2.3			X		
2.4	X				X

Inhaltlicher Bezug
Die nachfolgend ausgewiesenen Themenfelder sind die wesentliche inhaltliche Grundlage für die vorliegenden Aufgaben. Darüber hinaus können weitere, hier nicht explizit ausgewiesene Themenfelder für die Bearbeitung nachrangig bedeutsam sein.
Q1: Objektorientierte Softwareentwicklung
Q2: Datenbanksysteme
verbindliche Themenfelder: Objektorientierte Modellierung (Q1.1), Implementierung von Klassen und Assoziationen (Q1.2), Datenstrukturen (Q1.4), Konzeptionelle und logische Modellierung einer Datenbank (Q2.1), Datenabfrage und Datenmanipulation mit SQL (Q2.2), Kommunikation in Rechnernetzen (Q3.2),

II Lösungshinweise

In den nachfolgenden Lösungshinweisen sind alle wesentlichen Gesichtspunkte, die bei der Bearbeitung der einzelnen Aufgaben zu berücksichtigen sind, konkret genannt und diejenigen Lösungswege aufgezeigt, welche die Prüflinge erfahrungsgemäß einschlagen werden. Selbstverständlich sind jedoch Lösungswege, die von den vorgegebenen abweichen, aber als gleichwertig betrachtet werden können, ebenso zu akzeptieren.

Nicht für den Prüfling bestimmt

Aufg.	erwartete Leistungen	BE I	BE II	BE III
1.1	entwickeln, zeichnen EB-System Kunde anmelden Kontostand abfragen <<include>> überweisen Buchungen anzeigen Buchungen suchen beenden entwickeln zeichnen	2	3	
1.2	beschreiben Als Datenkapselung wird der kontrollierte Zugriff auf Attribute und Methoden von Klassen bezeichnet. Klassen sollen den internen Zustand anderer Klassen nicht in unkontrollierter Weise lesen oder ändern können. Eine Klasse hat eine Schnittstelle, die darüber bestimmt, wie mit der Klasse agiert werden kann. Die im Klassendiagramm enthaltenen Zugriffsrechte sind: public (+): Zugreifbar für alle Objekte. private (-): Zugreifbar nur innerhalb der eigenen Klasse. erläutern Die Attribute `pin` und `kontostand` der Klasse `Konto` sind `private` und somit von außerhalb der Klasse nicht sichtbar. Der Kontostand kann mit einer entsprechenden get-Methode abgefragt werden, eine set-Methode fehlt hingegen. Der Kontostand kann damit nur über die öffentlichen Methoden `einzahlen()` und `überweisen()` geändert werden. Das Attribut `pin` ist von außen ausschließlich über die Schnittstelle `login()` erreichbar.	2	2	
1.3	zeichnen : Konto iban = DE07123412341234123412 pin = **** kontostand = 599,48 : Buchung buTag = 18.04.2022 text = Einkauf Proper-Markt betrag = -42,70 : Buchung buTag = 10.04.2022 text = Geburtstagsgeld von Oma betrag = 100 : Buchung buTag = 31.03.2022 text = Lohn 03/2022 Soft GmbH betrag = 1108,48 : Buchung buTag = 01.04.2022 text = Mietzins 04/2022 betrag = -566,30	4		

Nicht für den Prüfling bestimmt

Aufg.	erwartete Leistungen	BE I	BE II	BE III
1.4	implementieren (Code siehe unten)	2	6	6

```
public class Konto {
  private String iban;
  private int pin;
  private double kontostand;
  private Buchung neueste;

  public Konto(String iban) {
    this.iban = iban;
    pin = EBVerwaltung.generierePin();
    kontostand = 0;
    neueste = null;
  }

  public boolean login(int pin) {
    return this.pin == pin;
  }

  public void einzahlen(String text, double betrag) {
    kontostand += betrag;
    hinzufuegenBuchung(new Buchung(text, betrag));
  }

  public boolean ueberweisen(Konto empfaenger, String text,
                             double betrag){
    boolean ok = false;
    if (betrag <= kontostand) {
      empfaenger.einzahlen(text, betrag);
      kontostand -= betrag;
      hinzufuegenBuchung(new Buchung(text, (betrag*-1)));
      ok = true;
    }
    return ok;
  }

  private void hinzufuegenBuchung(Buchung b) {
    b.setVorherige(neueste);
    neueste = b;
  }

  public String zeigeBuchungen() {
    String s = "Konto IBAN\t" + iban + "\tKontostand\t" +
       kontostand + " EUR\n" +
       "Buchungstag\tBuchungstext\tBetrag in EUR\n";
    Buchung b = neueste;
    while(b != null)  {
      s += b.toString()+ "\n";
      b = b.getVorherige();
    }
    return s;
  }

  public List<Buchung> sucheBuchungen(String begriff) {
    List<Buchung> gesuchte = new List<>();
    Buchung b = neueste;
    while(b != null) {
      if(b.getText().contains(begriff)) {
```

Aufg.	erwartete Leistungen	BE I	BE II	BE III

```java
        gesuchte.add(b);
      }
      b = b.getVorherige();
    }
    return gesuchte;
  }
}

public class Buchung {
  private Date buTag;
  private String text;
  private double betrag;
  private Buchung vorherige;

  public Buchung(String text, double betrag) {
    this.datum = new Date();
    this.text = text;
    this.betrag = betrag;
  }

  public String toString() {
    String s = buTag.toString() + "\t" + text + "\t" +
               betrag;
    return s;
  }
}
```

Aufg.	erwartete Leistungen	BE I	BE II	BE III
1.5	entwickeln		3	2

sucheKonto(iban:String)

- ok := pruefeIban(iban)
- kontoGesucht := null
- ok ?
  - J:
    - i := 0
    - solange i < konten.size() und kontoGesucht = null
      - k := konten.get(i)
      - iban = k.iban ?
        - J: kontoGesucht := k
        - N: i++
  - N: ∅
- Rückgabe: kontoGesucht

Aufg.	erwartete Leistungen	BE I	BE II	BE III
1.6.1	erläutern Das Client-Server-Modell beschreibt die Möglichkeit, Aufgaben und Dienstleistungen innerhalb eines Netzwerks zu verteilen. Die Aufgaben werden von Programmen erledigt, die in Clients und Server unterteilt werden. Der EB-Client fordert die Erledigung einer Aufgabe vom EB-Server an, wie das Anmelden und Durchführen von Überweisungen. Der Client ist aktiv. Der Server ist passiv und wartet auf Verbindungsanfragen. Er bearbeitet die Anfrage und gibt eine Rückmeldung über den Erfolg der Ausführung. Ein Dienst ist eine solche festgelegte Aufgabe, die der Server anbietet und der Client nutzt. Die Regeln der Kommunikation für einen Dienst (Kommandos, die Bedeutung der zwischen Server und Client ausgetauschten Daten und Formate) werden durch ein Protokoll festgelegt. Das Protokoll ist spezifisch für den jeweiligen Dienst.		4	
1.6.2	implementieren (Code siehe unten)	4	2	3

```
public class EBServer {
  private ServerSocket server = null;
  private int port;
  private EBVerwaltung eb;

  public EBServer(int port, EBVerwaltung eb){
    this.port = port;
    this.eb = eb;
  }

  public void startServer(){
    server = new ServerSocket(port);
    while(true){
      Socket sc = server.accept();
      sc.write("+OK E-Banking\n");
      String anforderung = sc.readLine();
      String[] werte = anforderung.split(";");
      Konto k = null;
      k = eb.anmelden(werte[1], Integer.parseInt(werte[2]));
      if(k == null) {
        sc.write("-ERR Login fehlgeschlagen!\n");
      } else {
        sc.write("+OK Willkommen\n");
        anforderung = sc.readLine();
        while (!anforderung.equals("quit")){
          werte = anforderung.split(";");
          Konto anderesKonto = eb.sucheKonto(werte[1]);
          if(anderesKonto != null ){
            if(eb.ueberweisen(k, anderesKonto, werte[2],
                              Double.parseDouble(werte[3])))
             sc.write("+OK Überweisung erfolgt\n");
            else
              sc.write("-ERR Überweisung nicht erfolgt\n");
          } else {
            sc.write("-ERR Ungültige IBAN\n");
          }
          sc.write("Aktueller Kontostand: " +
                   k.getKontostand() + " EURO\n");
          anforderung = sc.readLine();
        }
```

<table>
<tr><th rowspan="2">Aufg.</th><th rowspan="2">erwartete Leistungen</th><th colspan="3">BE</th></tr>
<tr><th>I</th><th>II</th><th>III</th></tr>
<tr><td></td><td><pre>   }
        sc.close();
      }
    }
}</pre></td><td></td><td></td><td></td></tr>
<tr><td>1.6.3</td><td>entwickeln, zeichnen

: EBClient
start()
<<create>>
Socket(hostname:String, port:int)
: Socket
connect()
{ ok:boolean}
opt
[ok = true]
readLine()
{ "+OK E-Banking" }
getAnmeldeDaten()
{ daten:String }
write(daten + "\n")
readLine()
{ antwort }
opt
[antwort = "+OK..."]
getUeberweisungsDaten()
{ daten:String }
loop [daten <> "quit"]
write(daten + "\n")
readLine()
{ ergebnis:String }
readLine()
{ kontostand…String }
getUeberweisungsDaten()
{ daten:String }
write("quit\n")
close()

entwickeln
zeichnen</td><td>
2</td><td>3</td><td>3</td></tr>
</table>

Aufg.	**erwartete Leistungen**	**BE**		
		I	**II**	**III**
1.7.1	beschreiben Die sequentielle Ausführung von Anweisungen eines Programms stellt einen Prozess dar, für den ein eigener Speicherbereich reserviert ist und der vom Betriebssystem verwaltet, gestartet und angehalten wird. Ein Thread ist ein einzelner in sich geschlossener Steuerfluss innerhalb eines Prozesses. Jeder Prozess besitzt einen Haupt-Thread, mit dem das Programm gestartet wird (main). Mehrere neue Threads können vom Programm gestartet werden. Diese Threads laufen dann alle quasi-parallel ab, besitzen jeweils einen eigenen Zustand mit Befehlszähler und Stack, arbeiten aber im Gegensatz zu Prozessen auf demselben Speicherbereich im Arbeitsspeicher.	2		
	erläutern Mehrere Threads können über die Kontomethoden auf die Variable `kontostand` eines Konto-Objekts gleichzeitig zugreifen. Es darf aber nicht vorkommen, dass ein Thread bereits aus einem Objekt liest, während ein anderer Thread noch Daten desselben Objekts ändert. Somit könnte eine Überweisung erfolgen obwohl keine ausreichende Deckung vorliegt.		1	
	benennen Das wird verhindert, indem die Methoden `ueberweisen()` und `einzahlen()` synchronisiert werden.		1	
1.7.2	entwickeln, zeichnen EBServer - port : int + EBServer(port : int, eb : EBVerwaltung) + startServer() Thread + run() + start() <<create>> 1 - eb EBVerwaltung 1 - bank EBThread + EBThread(sc : Socket, bank : EBVerwaltung) + run() 1 - sc Socket			
	entwickeln		1	1
	zeichnen	1		
	Summe 60	**19**	**26**	**15**

Nicht für den Prüfling bestimmt

Aufg.	erwartete Leistungen	BE I	BE II	BE III

2.1	überführen
	`Kunde(kid, nname, vname, strasse, plz, ort)`
	`Girokonto(iban, pin, kontostand, inhaber#)`
	`Buchung(bid, buTag, text, betrag, iban#)`
	`Darlehen(vertragsNr, datum, summe, rate, laufzeit, turnus, zinssatz, referenzkonto#)`
		3	1	

2.2.1	angeben
	`SELECT buTag, text, betrag`
	`FROM Buchung b`
	`WHERE iban = 'DE06550205000222222222'`
	`ORDER BY buTag DESC`
	`LIMIT 10;`
		3		

2.2.2	entwickeln
	`INSERT INTO Darlehen(vertragsNr, datum, summe, laufzeit, turnus, zinssatz, referenzkonto)`
	`VALUES(12345, NOW(), 30000, 10, 12, 1.76, 'DE06550205000222222222');`
	
	`UPDATE Darlehen`
	`SET rate = ROUND((summe * ((POWER(1+zinssatz/100,laufzeit)*zinssatz/100) / (POWER(1+zinssatz/100,laufzeit)-1)))/turnus, 2)`
	`WHERE vertragsNr = 12345;`
			2	3

2.2.3	entwickeln
	`SELECT kid, k.nname, k.vname, g.iban, d.vertragsNr, d.rate`
	`FROM Kunde k JOIN Girokonto g ON kid = inhaber`
	`JOIN Darlehen d ON g.iban = d.referenzkonto`
	`WHERE d.turnus = 12`
	`AND g.kontostand - d.rate < 0;`
	
	`UPDATE Girokonto g, Darlehen d`
	`SET g.kontostand = g.kontostand - d.rate`
	`WHERE d.referenzkonto = g.iban`
	`AND d.turnus = 12`
	`AND g.kontostand - d.rate >= 0;`
			4	2

2.2.4	implementieren
	`SELECT Kunde.*, SUM(summe) AS 'Kreditsumme aufgenommen'`
	`FROM Kunde JOIN Girokonto ON kid = inhaber`
	`JOIN Darlehen ON iban = referenzkonto`
	`GROUP BY kid`
	`HAVING SUM(summe) > 100000`
	`ORDER BY SUM(summe) DESC;`
			2	2

Aufg.	erwartete Leistungen	BE I	BE II	BE III
2.3	entwickeln, zeichnen Immobilie adresse iid art wert [1,n] abgesichert durch [1,m] Hypotheken-darlehen grundschuld eigenkapital [0,1] vertragsNr summe rate laufzeit Girokonto [1,1] referenziert [0,n] Darlehen is a [1,1] datum turnus zinssatz [0,1] Kleinkredit verwendungszweck [1,1] [1,n] besitzt [1,1] referenziert [0,n] stück Kunde [1,1] eröffnet [0,n] Depotkonto [0,m] enthält [1,n] Wertpapier wkn bezeichnung währung did datum [1,1] gehört zu [1,n] Transaktion art kurs tid datum anzahl			
	entwickeln		4	7
	zeichnen	2		

Aufg.	**erwartete Leistungen**	**BE I**	**BE II**	**BE III**
2.4	analysieren Die vorliegende Tabelle ist unnormalisiert. Die Werte in mehreren Spalten sind nicht atomar (z.B. die Felder der Spalten Uhrzeit und Kunde) und es treten Wiederholungsgruppen auf. Ein Beispiel für Wiederholungsgruppen ist die Spalte Thema. Problematisch ist das bei der Suche nach Datensätzen oder dem Sortieren nach bestimmten Kriterien. beschreiben Es treten Redundanzen auf, die zu Inkonsistenzen führen können. Wenn zum Beispiel die Adresse des Kunden Hans Hauser geändert werden soll, müsste das an mehreren Stellen gemacht werden (Änderungsanomalie). Eine Inkonsistenz tritt bereits auf. Die Telefonnummern von Herrn Hauser unterscheiden sich in den letzten beiden Stellen. Das Löschen von Datensätzen kann dazu führen, dass Daten verloren gehen, die eigentlich noch gebraucht werden (Löschanomalie). Wird zum Beispiel der Berater Kai Schlau gelöscht, weil er das Unternehmen verlässt, gehen auch die Kundendaten von Klara Kühn verloren. Das Einfügen neuer Kunden in die vorhandene Tabelle würde dazu führen, dass, so lange noch keine Beratungstermine vereinbart wurden, viele Datenfelder leer bleiben würden (Einfügeanomalie).	3	2	
	Summe 40	**11**	**15**	**14**

III Bewertung und Beurteilung

Die Bewertung und Beurteilung erfolgt unter Beachtung der nachfolgenden Vorgaben nach § 33 der Oberstufen- und Abiturverordnung (OAVO) in der jeweils geltenden Fassung. Bei der Bewertung und Beurteilung der sprachlichen Richtigkeit in der deutschen Sprache sind die Bestimmungen des § 9 Abs. 12 Satz 3 OAVO in Verbindung mit Anlage 9b anzuwenden.

Bei der Bewertung und Beurteilung der Übersetzungsleistung in den Fächern Latein und Altgriechisch sind die Bestimmungen des § 9 Abs. 14 OAVO in Verbindung mit Anlage 9c anzuwenden.

Der Fehlerindex ist nach Anlage 9b zu § 9 Abs. 12 OAVO zu berechnen. Für die Ermittlung der Punkte nach Anlage 9a zu § 9 Abs. 12 OAVO sowie Anlage 9c zu § 9 Abs. 14 OAVO wird jeweils der ganzzahlige nicht gerundete Prozentsatz bzw. Fehlerindex zugrunde gelegt.

Für die Bewertung in den modernen Fremdsprachen ist der „Erlass zur Bewertung und Beurteilung von schriftlichen Arbeiten in allen Grund- und Leistungskursen der neu beginnenden und fortgeführten modernen Fremdsprachen in der gymnasialen Oberstufe, dem beruflichen Gymnasium, dem Abendgymnasium und dem Hessenkolleg“ vom 7. August 2020 (ABl. S. 519) zugrunde zu legen. Demnach erfolgt die Bewertung und Beurteilung mit der Maßgabe, dass lediglich bei der Ermittlung des Prüfungsergebnisses (Note) aus Prüfungsteil 1 und 2 gerundet wird.

Darüber hinaus sind die Vorgaben der Erlasse „Hinweise zur Vorbereitung auf die schriftlichen Abiturprüfungen (Abiturerlass)“ und „Durchführungsbestimmungen zum Landesabitur“ in der für den Abiturjahrgang geltenden Fassung zu beachten.

Als Kriterien für die Bewertung und Beurteilung dienen unter Beachtung der Zielsetzung der gymnasialen Oberstufe nach § 1 Abs. 2 OAVO neben dem Inhaltlichen auch die in den Kerncurricula genannten überfachlichen Kompetenzen, insbesondere die Sprachkompetenz und Wissenschaftspropädeutik; dies zeigt sich u.a. in qualitativen Merkmalen wie Strukturierung, Differenziertheit, (fach-)sprachlicher Gestaltung und Schlüssigkeit der Argumentation.

Im Fach Praktische Informatik besteht die Prüfungsleistung aus der Bearbeitung eines Vorschlags, wofür insgesamt maximal 100 BE vergeben werden können. Ein Prüfungsergebnis von **5 Punkten (ausreichend)** setzt voraus, dass mindestens 45% der zu vergebenden BE erreicht werden. Ein Prüfungsergebnis von **11 Punkten (gut)** setzt voraus, dass mindestens 75% der zu vergebenden BE erreicht werden.

Gewichtung der Aufgaben und Zuordnung der Bewertungseinheiten zu den Anforderungsbereichen

Aufgabe	**Bewertungseinheiten in den Anforderungsbereichen**			**Summe**
	AFB I	**AFB II**	**AFB III**	
1	19	26	15	**60**
2	11	15	14	**40**
Summe	**30**	**41**	**29**	**100**

Die auf die Anforderungsbereiche verteilten Bewertungseinheiten innerhalb der Aufgaben sind als Richtwerte zu verstehen.